Using Reinforcement Learning to Improve Exploration Trajectories for Error Minimization

Thomas Kollar
Computer Science and AI Lab
The Stata Center, MIT
32 Vassar Street, 32-332
Cambridge, MA 02139
Email: tkollar@csail.mit.edu

Nicholas Roy
Computer Science and AI Lab
The Stata Center, MIT
32 Vassar Street, 32-332
Cambridge, MA 02139
Email: nickroy@mit.edu

***Abstract*— The mapping and localization problems have received considerable attention in robotics recently. The exploration problem that drives mapping has started to generate similar attention, as the ease of construction and quality of map is strongly dependent on the strategy used to acquire sensor data for the map.**

Most exploration strategies concentrate on selecting the next best measurement to take, trading off information gathering for regular relocalization. What has not been studied so far is the effect the robot controller has on the map quality while executing exploration plans. Certain kinds of robot motion (e.g, sharp turns) are hard to estimate correctly, and increase the likelihood of errors in the mapping process. We show how reinforcement learning can be used to generate good motion control while executing a simple information gathering exploration strategy. We show that the learned policy reduces the overall map uncertainty by reducing the amount of uncertainty generated by robot motion.

I. Introduction

Simultaneous localization and mapping (SLAM) is the problem of how to take noisy measurements of the environment, noisy estimates of sensor positions, and recover an estimate of the true state of the world. SLAM is fundamental to mobile robotics; an accurate and consistent model of the world is essential to ensuring the efficient and safe operation of robots in structured environments or where purposeful navigation is required. Under a set of common assumptions, in the limit of infinite time and observations, a perfect map can be attained [3]. However, the speed with which the estimated map converges to the true map is strongly dependent on how observations of the environment are taken and by the trajectory of the robot. Following the considerable attention that SLAM estimation techniques have received in the literature, SLAM-based exploration algorithms have been developed specifically to select how observations are taken and improve the convergence rate of the map estimation.

Most exploration algorithms compute the next best measurement to be taken (or in some cases the best sequence of measurements [10], [8]), where the "best" measurement or set of measurements typically are chosen to maximize the information gain with respect to the current map estimate. Once a measurement or sequence of measurements has been chosen, the task of executing the plan is given to some lower-level controller that tries to achieve the goal state or trajectory in minimum time or distance. However, the specific kinodynamic trajectory that a robot takes in executing exploration plans can have a large impact on the posterior map accuracy. Certain kinds of motion that are optimal for conventional, distance-optimal trajectories are undesirable when collecting data to build a map. In a holonomic robot, point turns are frequently employed before exploring in a new direction, but estimating the true angle of a fast point turn is difficult in comparison to a curved path with large radius of curvature. An exploration algorithm with many point turns increases the likelihood of errors in the resulting map.

Our method here will generate trajectories that minimize the posterior uncertainty of the robot position, independent of the measurements received during that trajectory. We expect this controller to be a good heuristic for generating trajectories that minimize map error. We will assume in this paper that we are using Extended Kalman Filter SLAM [14] with range and bearing measurements, although this formalism is not critical to the methods that we employ in that any measure of the expected error of the distribution would work.

The optimal solution to this problem might involve arbitrary trajectories with arbitrary turning rates. For our solution we approximately fix the translational velocity. Further, we fix the class of trajectories to be drawn from cubic splines. The final constraint is the assumption of a controller to follow the trajectory. The beautiful aspect of reinforcement learning is that we can plug in any controller and any robot model and optimize the trajectories according to this controller.

There does not unfortunately exist a computationally tractable way to optimize the posterior uncertainty of an extended robot trajectory (with or without integrating measurements); standard optimization techniques from the control literature will grow exponentially in complexity with the number of measurements. Our approach will therefore be to use reinforcement learning to find policies that minimize the posterior uncertainty given an ordered sequence of k future points that an exploration planner has selected for data gathering. Given this set of points, our algorithm can generate the best k step policy for visiting these points. The success of our approach depends on careful parameterization of the

action space. We use cubic splines (an example spline is in figure 1), which allows us to ensure that the derivative of the spline matches the current orientation of the robot at the start and end points.

We will show that our reinforcement learner successfully generates motion trajectories that minimize the posterior covariance of the robot in contrast to a standard hand-tuned controller that minimizes distance. We will also show that the posterior map learned from data collected by our policy learns more accurate maps. Finally, we will show that when the uncertainty generated by turning is high, then the learner will generate more curved paths. Conversely, when the uncertainty for translation is high, the learner generates trajectories that are close to the distance-optimal trajectory (see figure 1).

II. Simultaneous Localization and Mapping

The slam problem takes measurements of the environment and noisy estimates of sensor positions to recover an estimate of the true state of the world. Here, we assume a holonomic robot that has two types of measurements: landmark and location (odometry). A location measurement is the internal position measured directly from the wheel encoders in the form of a displacement $u_t = (\delta_{trans}, \delta_\theta)$. The landmark measurements z_t at time t include range r and bearing b from the robot to each of n landmarks, $z_t = (r_1, b_1, \ldots r_n, b_n)$. Examples of landmarks include stereo image features, corners and lines in a range scan. From the sequence of measurements $u_0, z_0, u_1, z_1, \ldots, u_t, z_t$, we wish to compute the robot pose (x, y, θ) and the (x_i, y_i) co-ordinates of each landmark, that is, the state $\xi = (x, y, \theta, x_1, y_1, \ldots, x_n, y_n)$.

Given that the measurements u_t and z_t are noisy, we cannot know the true state ξ_t. We can, however, compute a distribution over possible values of ξ_t using the Bayes' filter:

$$p(\xi_t|z_t, u_t, z_{t-1}, ut-1, \ldots) = \eta p(z_t|\xi_t) \int p(\xi_t|\xi_{t-1}, u_t)\, p(\xi_{t-1}, |z_{t-1}, u_{t-1}, \ldots) d\xi_{t-1}. \quad (1)$$

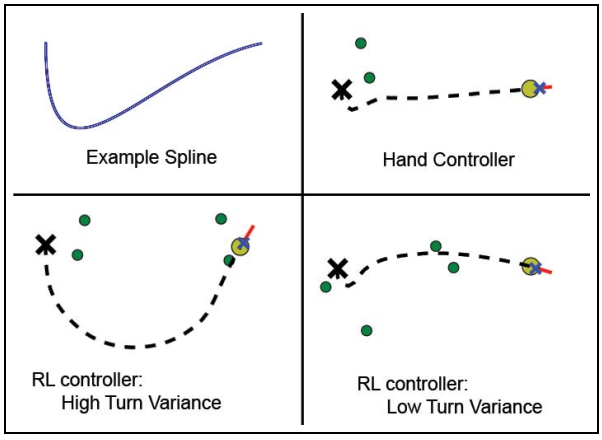


Fig. 1. Examples of controllers for different dynamics. Top left: is an example spline action. Bottom left: an example trajectory chosen by the controller when the error introduced by rotations is large. Top right: an example distance-optimal (hand) controller. Bottom right: an example trajectory chosen when there is little error introduced by rotations.

Note that $p(z_t|\xi_t)$ encapsulates our sensor model, $p(\xi_t|\xi_{t-1}, u_t)$ encapsulates our motion model and η is a normalization term.

The particular assumptions about the form of $p(\xi_t)$ and the mechanism for estimating this posterior is one of the major differences between different SLAM techniques. The Extended Kalman Filter provides an efficient technique for estimating $p(\xi_t)$ under the assumption that the posterior can be approximated reasonably well by a Gaussian distribution over ξ such that $p(\xi|z_{1\ldots t}, u_{1\ldots t}) = N(\mu_t, \Sigma_t)$. Additional assumptions are that the sensor model and motion model can be linearized and that the noise models are Gaussian; see [13] for a complete introduction to the Kalman filter and EKF. Under these assumptions, the expected error of the mean estimate can be quantified by a function of the covariance matrix such as the trace or determinant.

III. Reinforcement Learning (RL)

During exploration, a planner can only observe the current estimate of the state, rather than the true state of the world, which is a form of partial observability that often leads to computational intractability in finding good plans. However, we will assume a high-level planner has chosen waypoints for information gathering, and will focus on the problem of controlling the robot's uncertainty during travel between each waypoint. Because we can directly observe the state of the exploration process at all times, we can phrase this control problem as a Markov Decision Process (MDP). Briefly, an MDP is a tuple $\langle T, A, S, R, \gamma, s_0 \rangle$ where S is the state space, s_0 is the start state, A is the set of actions available to the agent, and $R(s, a, s')$ is the reward for taking action a in state s and ending in state s'. $p(s'|s, a) = T(s, a, s')$ is the probability of transitioning to state s' given a prior state s and action a. Finally $\gamma \in [0, 1]$ is a discount factor.

Standard techniques of solving MDPs include value and policy iteration. The optimal policy provides a mapping of states to actions such that the long-term expected reward of the policy is maximized. However, most practical techniques assume discrete state and action spaces and access to a complete description of the transition model T. In our exploration problem, the robot has a continuous state space (poses) and action space (motor commands). These issues are not necessarily fatal in that using intelligent discretization techniques for continuous state and action spaces have been used with success in other MDP settings. Most importantly, however, we do not have a closed form representation of the reward function, as the reward of each state and action is effectively a predicted EKF update from the robot pose for the specific action. Instead of solving for the optimal policy directly, we turn to reinforcement learning to map the current state to an appropriate action. We will show that our policy optimizes the multi-step robot trajectory and reduces the overall map uncertainty by reducing the amount of uncertainty generated by robot motion.

A. Policy Search by Dynamic Programming (PSDP)

Policy Search Dynamic Programming [1] learns policies by decomposing the reinforcement learning algorithm into a series of one-step policies. The algorithm operates by first learning a one-step policy at time T (e.g., the end time) by sampling states and actions, and learning to predict an action that maximizes the immediate one-step reward. A new one-step policy for the previous time step $T-1$ is then learned by sampling states and actions, obtaining an immediate reward as before. Using the policy previously obtained at time T, a two-step value is obtained for each state and action at $T-1$. Thus, the value associated with each state and action is accumulated from the immediate reward at time $T-1$ *and* the reward received by running the learned policy for time T.

The learner iterates at each time t generating a sample state $s^{(i)}$ and action a, propagating each sample $s^{(i)}$ forward through to time T using the policies π_i for $i \in t \dots T$. After sampling all of the actions in state s at time t, the best action $a^{'}$ is selected at state s. For each state s, there is thus a best action $a^{'}$. These are used as training pairs in a supervised learning problem at time t and the result is a classifier that, for any state s, provides an optimal action a. Bagnell et. al. give theoretical guarantees that the algorithm correctly solves the reinforcement learning problem.

The specific policy learner we use for each one-step policy is the multi-class Support Vector Machine [2]. This algorithm is a discriminative learner that assigns a label (i.e., optimal action) to each instance to be classified. We use an SVM to learn our one-step policy for two reasons: the SVM allows us to learn a classifier over an infinite input space, which allows us to use a continuous state space (x, y, θ, x_g, y_g), and the SVM is generally an efficient learner of large input spaces with a small number of samples. However, the SVM can only learn to label instances from a discrete set of labels; since we are associating optimal actions with each state, we must discretize our action space appropriately.

PSDP also requires that we have a proposal distribution from which to sample initial states. For our purposes here, we sample uniformly over the possible destinations. It should be noted that the PSDP algorithm produces a non-stationary deterministic policy, which is a more general class of policy than is usually used and PSDP is known to minimize approximation errors compared to value function techniques. The complete PSDP algorithm is given in figure 2.

PSDP(m_1, m_2, $\pi_{t+1,\dots T-1}$, *A)*

1) *for* $i = 1$ *to* m_1
   - *Sample* $s^{(i)}$ *uniformly over the size of the world*
   - $\forall a \in A$*:*
     - *Use* m_2 *Monte Carlo simulations to estimate the value of action* a_i *from* $s^{(i)}$*:* $V_{a_i, \pi_{t+1}, \dots, \pi_{T-1}}(s^{(i)})$
   - $\pi_t(s^{(i)}) = argmax_a(V)$
   - *Learn a multi-class SVM with* $s^{(i)}$ *as instances and* $\pi_t(s^{(i)})$ *as labels*
2) *Return the SVM as* π_t

Fig. 2. Policy Search by Dynamic Programming

IV. PSDP for Trajectory Control

The control problem we wish to solve is how to generate a motion trajectory for the robot from its current estimated pose (x, y, θ) to a destination position (x_g, y_g) (or sequence of destinations, $(x_g^0, y_g^0, x_g^1, y_g^1, \dots, x_g^n, y_g^n)$. We therefore need to define a state space, action space and reward function.

A. State Space

The state space of the controller is the current robot pose (x, y, θ) and a sequence of destinations $(x_g^0, y_g^0, x_g^1, y_g^1, \dots, x_g^n, y_g^n)$ chosen by some high-level exploration planner to maximize information gain[1]. Note that by our choice of PSDP, we will decompose the larger reinforcement learning problem into a series of horizon-one problems and rely on PSDP to learn a controller that generates trajectories that move smoothly across a sequence of waypoints.

B. Action Space

In order to express motion between waypoints as a one-step learning problem, we cannot use direct motor commands as the actions. We instead use as the action space the simplest polynomial that both interpolates the start and end goal and allows us to constrain the start orientation. This is the space is of cubic splines between the current location of the robot and the sequence of information waypoints. (A cubic spline consists of a set of cubic polynomials interpolating a set of control points.) The spline is fixed such that the derivative of the spline at the start control point matches the orientation of the robot at its current position. The orientation with which the robot arrives at the goal point and the magnitude at the start and end locations are not fixed. The spline is parameterized as follows ($t \in [0, 1]$):

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \tag{2}$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \tag{3}$$

Using this parameterization and providing values for the x and y positions $(x, y)|_{t=0}$, $(x, y)|_{t=1}$ and derivatives $(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t})|_{t=0})$, and $(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t})|_{t=1})$ at $t = 0$ and $t = 1$, one can solve for the specific constants. Although there are nominally eight total parameters, five of these parameters are fixed. $(x, y)|_{t=0}$ is the start position of the robot and $(x, y)|_{t=1}$ is the destination position (x_g, y_g). Further, we maintain the orientation constraint of the robot by requiring that

$$\frac{\frac{\partial y}{\partial t}|_{t=0}}{\frac{\partial x}{\partial t}|_{t=0}} = \tan(\theta). \tag{4}$$

Thus, we have one free parameter at the start point (the magnitude of the derivative at $t = 0$) and two free parameters

[1]For the purposes of clarity, and without loss of generality, we describe the system in Euclidean co-ordinates. In practice, we implemented the state representation in polar co-ordinates in the robot frame of reference, reducing the number of variables required to express the joint robot pose and goal position.

(the orientation of arrival and the magnitude of this ratio) at the destination.

The kinematic trajectory of the spline does not directly provide motor control commands. To convert the spline into an actual motor sequence, we use a simple $P - D$ control loop on the current robot position estimate and a point on the spline a distance d in front of the robot.

C. State transitions

Assuming that we have an ordered set of n points p_t (the current state) that we want to visit at time t: $\langle p_t, p_{t+1} \dots p_{t+n} \rangle$. Our new set of points (the new state) will be $\langle p_{t+1}, p_{t+2} \dots p_{t+n}, p_{t+n+1} \rangle$, where p_{t+n+1} is chosen uniformly over some maximum distance that we expect to travel. In situations where the high-level exploration planner were choosing information waypoints in some predictable manner, the state transitions might not necessarily be uniform. However, this is fully transparent to PSDP and reinforcement learning techniques in general.

D. Reward

The reward for each action is computed from the change in uncertainty of the posterior robot pose estimate. If $u_{t=k}$ is the original uncertainty and $u_{t=k+1}$ is the final uncertainty, then $r_{t=k} = u_{t=k} - u_{t=k+1}$, encouraging the policy to introduce as little uncertainty as possible through the motion of the robot. Since we are using an Extended Kalman filter, the posterior is given by a Gaussian with covariance Σ and mean μ, and a good measure of uncertainty in a Gaussian distribution is the trace of the covariance matrix; the use of the trace (in contrast to the determinant) for information gathering in exploration problems has been shown elsewhere to improve performance [8]. While any matrix norm may work as a metric, the trace corresponds intuitively to the sum of the magnitudes of the principal components of the Gaussian distribution. Similarly, the determinant corresponds approximately to the volume.

We have not specifically incorporated obstacle avoidance into the trajectory generation, but in order to model the cost of potential collisions, we would provide a penalty term for leaving a corridor and a penalty term for running into obstacles. In particular, we would define a corridor of width r on a straight line between (x, y) and (x_g, y_g) that is assumed to be free of obstacles. In moving from (x, y, θ) to (x_g, y_g) the robot would be penalized with a very large negative reward if at any instant it moves further than w away from the straight-line vector between the start and goal points. This obstacle avoidance measure would model the effect of corridor environments; note that we could use the obstacles already in our map to generate large punishment. We could also replan immediately upon discovering new obstacles.

E. Motion Model

In addition to the components required by PSDP to learn good controllers, we also require a motion model of the robot. This motion model is the principal reason for using intelligent control to minimize uncertainty, but the model is not used by the learner; the learner instead uses data to learn how to minimize the uncertainty captured by the model. We use the motion model to simulate the outcome of robot motion and to update the EKF posterior, but do not give knowledge of the robot motion to the reinforcement learning algorithm.

We use a common motion model that represents each motion as a combination of a translation and a rotation,

$$\delta_{trans}^{'} = \delta_{trans} + N(0, \delta_{trans}\sigma_{trans}) \quad (5)$$

$$\delta_{\theta}^{'} = \delta_{\theta} + N(0, \delta_{\theta}\sigma_{\theta}), \quad (6)$$

where each action has some (unobserved) noise added according to normal distribution. In the above, δ_{trans} is the commanded translation from the previous location and δ_θ is the commanded rotation from the previous location, and the standard deviation of each noise term is proportional to the commanded translation or rotation. A similar but more general motion model would include systematic bias terms in the induced noise [4].

From the translation and rotation, we can compute the new position of the robot as

$$x' = x + \delta'_{trans} cos(\theta) \quad (7)$$

$$y' = y + \delta'_{trans} sin(\theta) \quad (8)$$

$$\theta' = \theta + \delta'_{\theta}. \quad (9)$$

This model is equivalent to saying that the robot first moved directly forward by $\delta_{trans} + N(0, \delta_{trans}\sigma_{trans})$ and then turned by $\delta_\theta + N(0, \delta_\theta\sigma_\theta)$.

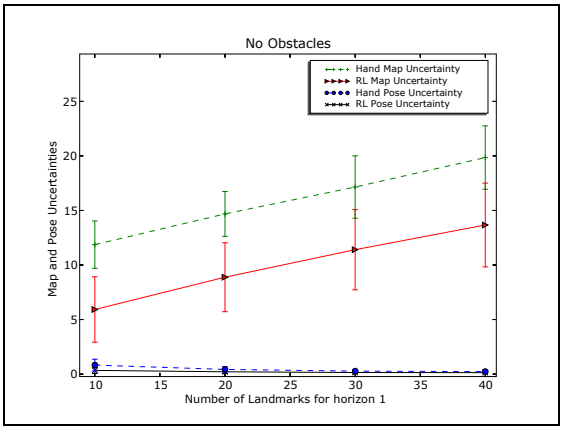
Note that good techniques exist for learning robot motion models from data [4], which allows us to perform the learning process offline in simulation. The n SVMs corresponding to the policy of length n can then be used directly on a physical robot.

V. Experiments

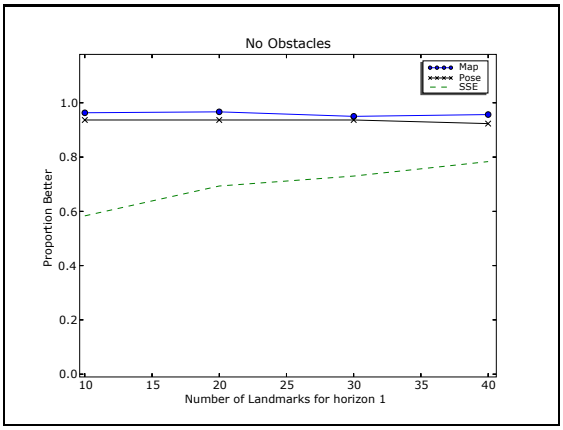
We used reinforcement learning to learn the best trajectory for a robot tracking its position with an Extended Kalman filter in a world of size 35×35 with point features. We used Policy Search Dynamic Programming to learn multi-step policies; each one-step policy was learned using a support vector machine with a Gaussian kernel, $C = 5$, and $\gamma = 0.2$. These parameters were computed by doing a parameter search with one step input/output pairs. Each learning problem used $m_1 = 1,000$ samples were taken with $m2 = 1$ roll-out samples for each start sample. A total of 160 actions for each sample were considered by sampling from the space of possible cubic splines. Both the landmark locations and the goal positions were chosen at random. Training each horizon-one problem with 1,000 samples takes about an hour. The time to classify actions using an SVM takes milliseconds, making it viable for use in online SLAM.

A. One-step policies

Looking at figure 3(a), we can see the improvement over the distance-minimizing hand controller in terms of the overall



(a) Map and Pose Uncertainties



(b) Proportion of times RL is better than the Hand-controller

Fig. 3. A comparison over all destinations of the RL controller and the hand controller (a) in terms of pose and map uncertainty and (b) in terms of the proportion of time that the RL controller is better for various numbers of randomly selected landmarks.

map and pose uncertainties. In figure 3(b), it is clear that the map and pose uncertainties are better than the hand controller nearly 98% of the time. Further, up to 80% of the time the RL controller will obtain better overall sum squared error in the maps. Note that figure 3 reflects destinations at least 7 meters from the current location of the robot and randomly selected landmarks.

These results show that minimizing the one-step pose uncertainty indeed provides trajectories that will help to minimize the total map uncertainty as well as the sum squared error. Further, as the number of landmarks increases, the proportion of times the sum squared error is better increases. We speculate this is because the position of the robot has fewer jumps in it, thereby enabling the controller to generate a path that more closely resembles the target spline. Note that the average uncertainty of the map increases with the number of landmarks, because more landmarks are distributed further from the robot, contributing to the overall uncertainty. The salient feature of figure 3 is the difference between controllers.

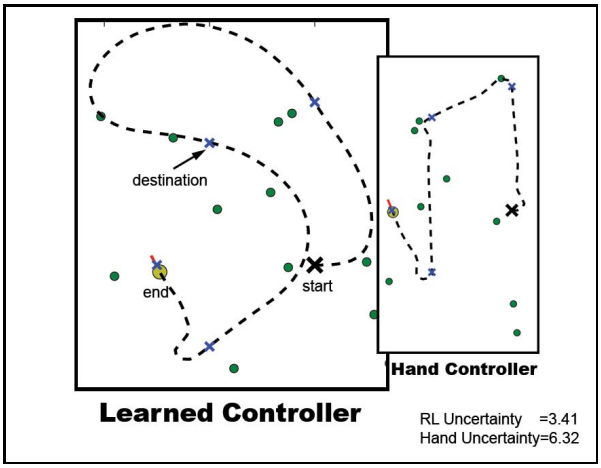


Fig. 4. Exploration trajectories: a sequence of 4 exploration points are given by a high-level planner for exploration. On the left is the true four-step trajectory generated from the learned controller. On the right is the trajectory of the distance-minimizing controller.

Note from figure 1 that when the variance in the turn error is very low, the trajectory will converge to the distance optimal trajectory.

B. Multi-step policies

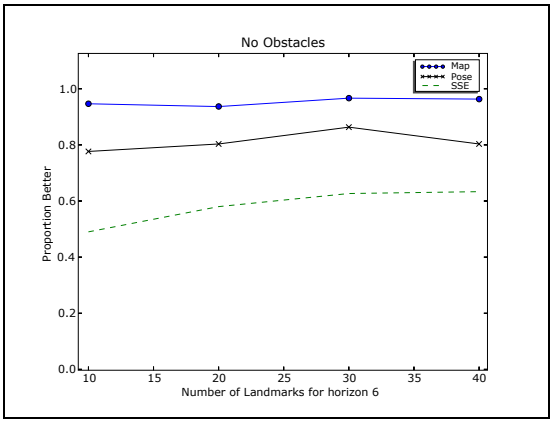
The creation of multi-step policies takes polynomial time in the number of steps that we want to optimize. In particular, the number of simulation steps that we have to perform is $O(\frac{1}{2}n(n+1)akv)$, for n the horizon, a the number of actions, m_1 the number of samples at a given timestep and m_2 the number of approximations of the value function that we need to perform.

Since the computation of the t^{th} policy takes t^2 simulation steps for a fixed problem, the the bottleneck of generating a multistep policy resides in the training. Computing a one-step policy corresponds to simply setting $t = 1$, and therefore is significantly faster than generating a true multi-step policy. In our experiments, the total training time for a one-step policy is about 30 minutes, while a four-step policy can take 5 hours. The training of each SVM is trivial; in our experiments it takes well less than a minute per SVM. Finally, when we need to evaluate a state (perform classification using an SVM), the computational time remains at milliseconds, making it viable for use in online SLAM.

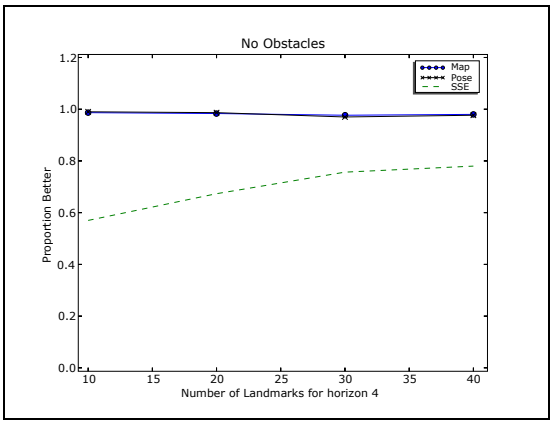
In figure 5, we see the result of learning multi-step policies versus one-step policies. Clearly the one-step policies perform significantly worse than the multi-step policies. The reason for this is that there is no lookahead. For example, the two step policy can anticipate the need to be at the first destination at a certain angle, where the one-step policy is oblivious to this need.

C. Multistep Policy Evaluation

As a proxy for investing the computational time into creating an n step policy for an n step problem, we decided to compare the benefits of using a true four-step policy over



(a) Iterated one-step policy for horizon 6



(b) True four-step policy for horizon 4

Fig. 5. Comparison of a (a) one-step policy run for 6 destinations and (b) a true 4-step policy.

repeating the one-step policy six times. Further, we perform a comparison directly to the hand controller for each set of landmarks and destinations. Figure V-B shows an example trajectory chosen by the controller, and the corresponding trajectory generated by the hand controller. In this example, the uncertainty of the map learned by the RL controller compared favorably to the map learned by the hand controller.

In figure 5(a) we used a 1 step policy over 6 steps and compared this to the hand controller. We can see a reduction in the sum squared error some of the time, the pose uncertainty about 80% of the time and the map uncertainty almost all of the time. This shows that a one-step controller can improve overall uncertainty somewhat.

For the true four-step policy, we performed a quantitative comparison of the learned policies and hand-crafted controllers. We used $m_1 = 1000$ samples and we test on 300 trials consisting of four steps each, where actions are chosen either by the hand controller or by the learned RL policy. Here we used a true four step policy and not a four step policy that is generated via chaining four one-step policies. The results are presented in figure 5(b). Clearly, the RL controller is doing better than the hand controller.

Comparing figure 5(a) and 5(b) we see that the true 4 step policy results in a reduced map uncertainty nearly all of the time and lower sum squared error in the map up to 80% of the time. In contrast to the iterated one-step policy, we notice that the true 4-step policy significantly outperforms the iterated 1-step policy.

These results indicate that the SLAM problem can greatly benefit from the use of trajectory planning as we have presented it here. In particular, when taking into account the next n destinations, the RL will produce a policy much better than a standard hand tuned policy.

VI. Conclusions

We have shown a novel approach to trajectory control for a mobile robot performing exploration. Given destination points for information gathering, we have shown that reinforcement learning can be used to learn a control policy that finds actions to minimize the increase the posterior uncertainty of the robot pose. We have furthermore shown that this is a good heuristic for generating better maps. Using this technique we can generate arbitrary k-step policies that can optimize the full trajectory given knowledge of k destinations. While in the future real data will be obtained from real environments to ensure that our simulations are accurate, these results show ways in which the robot can create better maps by taking different actions.

References

[1] J.A. Bagnell, S. Kakade, A. Ng, J Schneider, "Policy search by dynamic programming," *Neural Information Processing Systems*, MIT Press, Vol. 16, December, 2003.

[2] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*: Cambridge University Press, Cambridge, UK. 2000.

[3] M.W.M.G. Dissanayke, P.M. Newman, H. F. Durrant-Whyte, S.Clark, and M.Csorba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, 2001.

[4] Eliazar A.I. and Parr R. "Learning Probabilistic Motion Models for Mobile Robots," *Proceedings of the Twenty First International Conference on Machine Learning*, 2004.

[5] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, Vol 101, pp. 99-134, 1998.

[6] M. Lagoudakis and R. Parr, "Least-Squares Policy Iteration," *Journal of Machine Learning Research*: Vol. 4, p. 1107-1149, 2003.

[7] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*: J. Wiley and Sons, 1994.

[8] R. Sim and N. Roy, "Global A-Optimal Robot Exploration in SLAM" Proceedings of the IEEE/RSJ International Conference of Robotics and Automation (ICRA), 2005.

[9] R. Sim, G. Dudek, N. Roy, "Online Control Policy Optimization for Minimizing Map Uncertainty During Exploration" Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004). New Orleans, April 2004. pp. 1758 - 1763.

[10] C. Stachniss and G. Grisetti and W. Burgard. "Information Gain-based Exploration Using Rao-Blackwellized Particle Filters," *Proc. of Robotics: Science and Systems (RSS)*, Cambridge, MA, USA, 2005.

[11] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*: MIT Press, Cambridge, MA, 1998.

[12] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*: MIT Press, Cambridge, MA, 2005.

[13] G. Welch and G. Bishop, "An Introduction to the Kalman Filter." Technical Report 95-041, UNC Chapel Hill.

[14] J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, pages 1442–1447, Osaka, Japan, November 1991.